Falcon 云应用开发规范

一、总体介绍

Falcon 利用 Memcache、Nginx 反向代理等基础设施，提供易于扩展和使用的云服务能力。由于单台服务器的服务能力是有限的，因此我们设计了多台 Tomcat 作为应用服务器，共同组成了一个集群，向最终用户提供高可用性、可靠性和一致性的 Web 服务能力。
为了达到这个目标，按照 Web 应用的特点，Falcon 设计和提供了下图所示的部署结构：

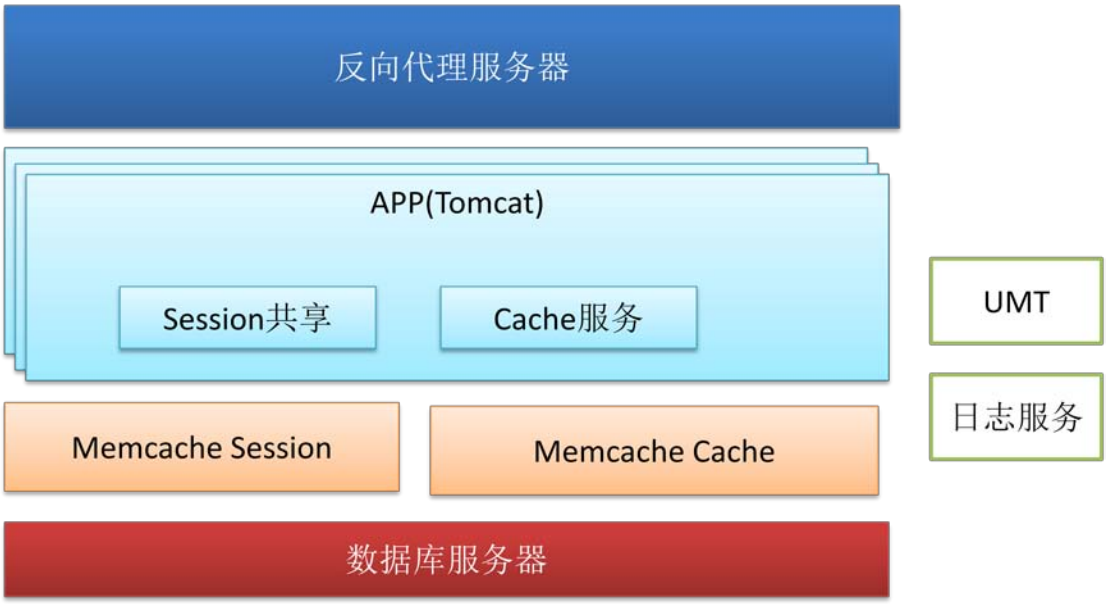


图 1. Falcon 部署结构图

这其中各个组成部分的作用是：

1. Falcon 组件:

☐ 应用服务器（Tomcat）：通过将应用部署在多台的 Tomcat 上，我们可以减少单点故障的可能性、同时获得了并平行扩展系统服务容量的能力。通过将应用部署在多台的 Tomcat 上，我们可以减少单点故障的可能性、同时获得了并平行扩展系统服务容量的能力。

☐ 反向代理服务器：在这个结构中，反向代理起到的是多台 Tomcat 服务的整合功能。这样用户就可以通过同一个入口 IP/域名，访问 Web 服务。在我们实际的部署中，这个使用的是 NGINX 服务器。

☐ Memcahe Session 服务器：单独部署的 Memcahe 集群，为 Tomcat 的 session 提供缓存服务。通过与部署在 Tomcat 中的 session 共享组件，完成多机 Session 信息的共享。从而实现用户 Session 状态的自由迁移。

☐ Memcache Cache 服务器：单独部署的 Memcache 集群，为应用中的数据库数据提供统一的缓存服务，维护多机中应用状态的一致性。

☐ MySql：数据库服务器，提供基本的数据库访问服务。

□ Session 共享组件：配置部署于 Tomcat 容器中的组件，能够自动的将 Session 中的信息缓存到 memcache 中，提供多机 Session 信息共享。
□ Cache 服务组件：被应用代码调用，提供统一的缓存服务。

2.基础服务

□ UMT：提供 Web 系统的单点登录功能。目前科技网使用 Passport（ https://passport.escience.cn ）就是该应用的一个部署实例。
□ 日志服务：提供简单的应用日志服务，提供 RESTful 接口的 API 访问，方便应用对日志信息的分析。

二、API 接口

按照第一部分所描述的部署结构，需要对开发的应用代码进行一定的改造。这部分的功能需要在应用中加入 falcon-api-0.0.3-snapshot.jar。下面将按模块说明改造的方法：

1.程序共享内存数据

为了实现程序的动态扩展，应用程序有一些数据需要所有的机器都可以访问，因此这类数据需要找一个集中的地方来存储。因此使用共享内存数据服务是一个比较好的选择。Falcon 为开发者提供了这样接口
(net.duckling.falcon.api.cache.ICacheService)，其定义如下：

Falcon CachService	

ICacheService	/**
	* @description 通过关键字获得可序列化对象
	*/
	Object get(String key);
	/**
	* @description 将可序列化对象放入缓存中
	*/
	void set(String key, Object value);
	/**
	* @description 从缓存中删除该关键字对应的条目
	*/
	void remove(String key);
	/**
	* @description 缓存初始化服务，建立连接
	*/
	void doInit();
	/**
	* @description 缓存关闭服务，断开连接
	*/
	void doDestroy();

当需要使用 CacheService 时，必须先初始化 ICacheService。推荐使用 Spring 框架来完成此过程，示例如下：

Spring 初始化（使用 Memcached 实现）

```
<bean id="cacheService"


class="net.duckling.falcon.api.cache.impl.MemcachedCacheService"

     init-method="doInit" destroy-method="doDestroy">

   <property name="memcachedURL" value="127.0.0.1:11211"/>

</bean>
```

当然，你可以通过 Java 代码将这个服务初始化

Java 代码初始化（使用 Memcached 实现）

```
import net.duckling.falcon.api.cache.impl.MemcachedCacheService;

import net.duckling.falcon.api.cache.ICacheService;

//…

ICacheService cacheSerivce = new MemcachedCacheService();

cacheService.setMemcachedURL(“127.0.0.1:11211,192.168.0.1:11211”);

//初始化连接

cacheService.doInit();

//调用写入和读取函数

//cacheService.set(“greeting”, “Hello!”);

//完成以后调用 doDestroy 清理 memcahed 服务器连接

cacheService.doDestroy();
```

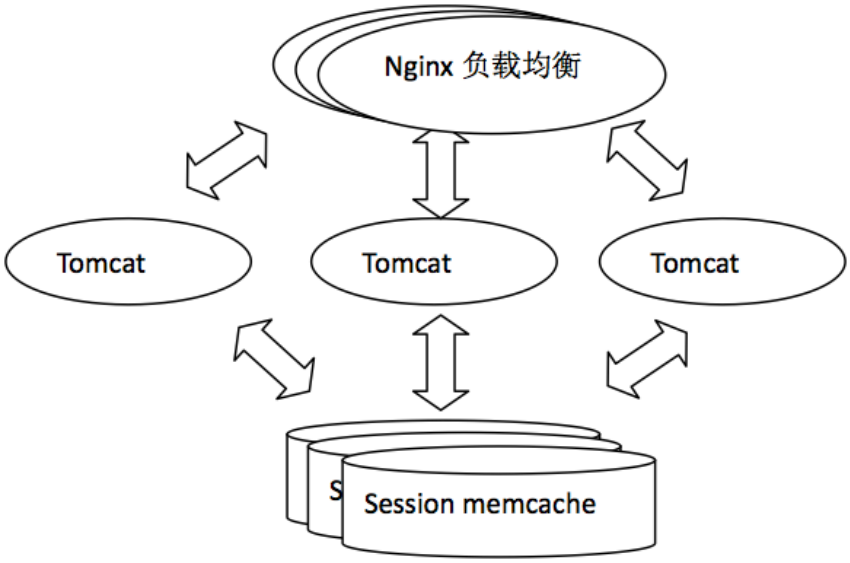
注意：任何存储放在 CacheService 中的 Object 都需要实现 Serializable 接口。

2.会话共享

为了实现多台服务器服务能力的横向扩展能力,需要在这多个服务器间共享 Session。 以实现在单个服务器宕机的情况下,可自动将用户的服务迁移到其他可用的服务器上。目前 Falcon 提供了两种容器下的 Session 共享服务:Tomcat 下的 Session 共享服务,Apache 的 PHP 容器下的 Session 共享服务。版本要求：

Java 平台:jdk 1.7,tomcat 7.0.26；

Tomcat 下的 Session 共享
在 Falcon 的方案中,Session 共享的实现是通过将 Session 的数据存储到中心的 memcache 服务器上实现的。我们建议的部署方案如下图所示:



在这样的部署结构下,一旦其中的一个 Tomcat 服务器宕机了,用户会被调度到其他的 Tomcat 上,而用户的 session 信息也可以从 Memcache 中获取。从而保证了整个服务流程不 会因为这样的原因中断。Tomcat 下配置 Session 共享服务请按照下面提到的步骤进行:

2.1.1. 从 FTP 上下载配置好的 tomcat 服务器

falcon 已经将 session 共享的 tomcat 放在了 FTP 上, 部署时从 http://ftp.cerc.cnic.cn/pub/project/deployment/apache - tomcat - 7.0.26_msm.zip 下载后,解压到你想部署的路径上。

2.1.2. 配置 memcache 服务器

假设${path_to_tomcat}是你刚才解压的 Tomcat 的根路径。这时你需要打开 ${path_to_tomcat}/conf/context.xml。在这个文件里你可以看到:

会话共享配置

```
<Manager

className="de.javakaffee.web.msm.MemcachedBackupSessionManager"

sticky="true"

memcachedNodes="n1:10.10.1.64:11211"

failoverNodes=""

requestUriIgnorePattern=".*\.(png|gif|jpg|css|js|ico)$"
sessionBackupAsync="false"

sessionBackupTimeout="100"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoT
ranscoderFactory"

customConverter="de.javakaffee.web.msm.serializer.kryo.JodaDateTime
Registration,de.javakaffee.web.msm.serializer.kryo.WicketSerializer
Factory,net.duckling.serializer.CustomKryoRegistration"

/>
```

在实际的配置中,将 memcacheNodes 配置修改成你实际部署的 memcache 的服务器的 地址即可。如果是配置多个节点,则只需在多个节点之间使用空格分开。例如: n1:192.168.0.84:11211 n2:192.168.0.84:11213 即可!

2.1.3. 开发要求

实际的使用中,按照一般的 Java Web 程序的对 Session 的使用方法使用即可。但由于需 要将 Session 的内容复制到了 memcache 服务器上,这个过程是涉及到多个服务间的操作。 因此这个过程要求数据能够序列化。所以对使用 Session 共享服务的应用,有两个条件:

存入 Session 中的数据必须是可序列化的(实现 java.io.Serializable 接口),以保证程序能在多个服务器间复制数据。

由于 Session 是存放在中心的服务器上的,涉及到了网络调用。因此,尽量不要将无关的数据存入 Session。

3.用户注册和登录

UMT的认证登录的配置和开发请参见http://passport.escience.cn/help_oauth.jsp页获取所需的信息。

三、部署例子

使用 Falcon 作为基础开发的软件，需要按照第一节所示的结构去部署。附件《UMT 部署文档》是以 CSTNET 的用户管理工具为例进行介绍的安装过程。由于 UMT 的特殊性，请略去服务器证书申请的部分，普通的服务将不需要使用 HTTPS 作为服务的协议。同时请酌情对 NGINX 反向代理部分进行一定的修改。